Broose: A Loose Distributed HashtableBased on the De-Brujin Topology

Anh-Tuan Gai, Laurent Viennot

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Broose: A Loose Distributed Hashtable Based on the De-Brujin Topology

Anh-Tuan Gai — Laurent Viennot

N° 5147

Mars 2004

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--5147--FR+ENG

Broose: A Loose Distributed Hashtable Based on the De-Brujin Topology

Anh-Tuan Gai*† , Laurent Viennot*†

Thème 1 — Réseaux et systèmes
Projet Gyroweb

Rapport de recherche n° 5147 — Mars 2004 — 10 pages

Abstract: We describe a peer-to-peer system similar to Kademlia based on the De-Brujin topology with provable consistency and allowing every message exchange to reinforce contact information. Ultimately simple routing tables are achieved by maintaining only two "buckets" of neighbors and providing maximal accuracy of routing tables.

Key-words: peer-to-peer, distributed hashtable, De-Brujin

* INRIA Rocquencourt, Domaine de Voluceau, 78153 Le Chesnay, France, {Anh-Tuan.Gai, Laurent.Viennot}@inria.fr

† Travail financé par le projet PairAPair de l'ACI Masse de données

Broose : une table de hachage distribuée souple basée sur le graphe de De-Brujin

Résumé : Un protocole pair-à-pair similaire à Kademlia est proposé sur la base d'une topologie inspirée du graphe de De-Brujin. Les tables de routage reposent sur la maintenance de deux pools d'adresses de contacts seulement. Les échanges de messages permettent de mettre à jour ces pools.

Mots-clés : pair-à-pair, table de hachage distribuée, De-Brujin

1 Introduction

This paper describes Broose, a peer-to-peer distributed hashtable protocol. It allows to store key, value associations in a loose way in the sense that nodes in charge of storing the values associated to a given key are defined in a loose way conversely to most previous distributed hashtables propositions. This feature is mostly inspired from Kademlia [4]. The major improvement over Kademlia resides in the use of a De-Brujin Topology for reducing the routing table size and enhancing its accuracy.

Broose follows the basic approach of many distributed hashtables where keys are 160-bit binary strings and where each node randomly chooses an ID in the 160-bit key space. As in Kademlia, a key, value association is stored on the nodes close to the key for some metric instead of splitting the key space among nodes in a rigid manner. This allows provable consistency even in a highly dynamic asynchronous context. Instead of storing contact information about nodes in charge of specific parts of the key space, buckets of possible contacts are maintained with the goal of achieving best accuracy. Contrarily to Kademlia that maintains $O(\log N)$ buckets for each node where N is the number of nodes, Broose maintains only two buckets per node allowing the protocol to concentrate on achieving best accuracy of them.

To allow this, the protocol is based on the De-Brujin topology. Although this topology is intrisicly asymmetric, the protocol allows efficient update of the buckets of contact information by using simultaneously and indifferently two reverse lookup schemes. A lookup allows any node to locate nodes with IDs close to a destination key by following simultaneously α paths.

Section 2 describes the De-Brujin topology and makes an overview of existing distributed hashtables solutions based on it. Section 3 describes the breakthrough proposed by Kademlia. Section 4 describes the main details of Broose for adapting the De-Brujin topology to the ambitious goals of Kademlia design.

2 De-Brujin topologies

2.1 The De-Brujin Graph

For any n-bit binary string u, let $u[i,j]$ denote the substring from the ith bit to the jth bit. For example, $u = u[1,n]$. The De-Brujin B_n graph (see [3] for an overview) over $N = 2^n$ nodes is defined as follows. Every node u has two successors: $u[2,n]0$ and $u[2,n]1$ (one of these successors may be u itself). This simply defined graph has the property of having constant in-degree and out-degree 2 and logarithmic diameter n. One can easily find a route from u to any node $v = v_1 \cdots v_n$: $u[2,n]v_1 \rightarrow u[3,n]v_1v_2 \rightarrow \cdots \rightarrow u[n,n]v_1 \cdots v_{n-1} \rightarrow v$. Note that another route can be similarly found by following edges backward. This graph can be easily generalized to a $\Delta = 2^b$ degree graph with $\log N / \log \Delta$ diameter.

2.2 De-Brujin Distributed Hashtables Overview

Several propositions [1, 5, 2] have been made to adapt the De-Brujin topology in the context of distributed hashtables. They address the problem of constructing a De-Brujin like topology with a variable number of nodes N. They share some similarities: each node is supposed to be identified by an n-bit binary string u (typically $n = 160$) and is supposed to store all key, values associations for some range of keys I_u. I_u is almost always an interval containing u. For the sake of simplicity, we will identify a node u and its ID.

Koorde [2] defines I_u as the keys smaller than u and greater than the preceding ID. (All IDs are ordered along the cycle of unsigned integers of n bits modulo 2^n.) For Naor and Wieder [5], node u must store the keys between u and the next ID. They view the keys and the IDs as the binary representations of reals in $[0, 1)$ (u is associated to $0.u_1 \cdots u_n$) but obtain a very similar definition of I_u. (This "continuous approach" allows a very nice interpretation of the De-Brujin graph where the two predecessors of a node with real ID r are $r/2$ and $r/2 + 1/2$.) In D2B [1], each node u has a label $u[1, l]$ which is a prefix of u. l is the smallest index such that the label of u is unique. I_u is then defined as the keys k admitting this label as prefix ($k[1, l] = u[1, l]$).

We can define $I_u << 1$ as the set of keys k such that $0k[1, n-1]$ or $1k[1, n-1]$ is in I_u. The basic idea in the three propositions consist in linking node u to all nodes in charge of any key in $I_u << 1$. Note that u may have more than 2 successors when the IDs are not smoothly balanced. Naor and Wieder introduce the "smoothness" ρ of the IDs distribution as $\rho = \max_{u,v} \frac{|I_u|}{|I_v|}$. All nodes have thus degree $O(\rho)$, however it can be shown that the average degree is constant. Koorde maintains a constant degree by pointing only towards the first node whose interval intersects $I_u << 1$. Each node also points to the node with next ID allowing to find all nodes whose interval intersects $I_u << 1$ linearly. This results in a $O(\log N + \rho)$ routing procedure instead of $O(\log N + \log \rho)$ as achieved by the two other propositions.

This smoothness factor ρ is interesting for analyzing the reliability of the protocol towards node insertion and deletion. If the IDs are chosen randomly, it is a classical result to show that $\rho = O(\log^2 N)$.

Finally, the above propositions can easily be generalized by shifting b at a time to allow $O(\log N/b)$ lookups at the cost of expected degree $O(2^b)$.

3 Kademlia : a Loose Distributed Hashtable

3.1 Kademlia

Kademlia [4] is a distributed hashtable architecture similar to Pastry [6] allowing a lot of flexibility with regard to the topology compared to other schemes such as Chord [7] or the De-Brujin schemes [5, 1, 2] described above. The basic idea is to use a metric such that any node at distance d from some destination stores in its routing table nodes at distance

roughly $d/2$. Kademlia uses the xor metric indicating that two IDs are close if they share a long common prefix.

The main breakthrough of Kademlia is to loosen the rule defining which nodes are in charge of storing values for some key. This is achieved along with another design goal which is the rapid population of routing tables. For that reason, a node performing a request successively contacts all nodes on the route towards the destination instead of using routing. The basic idea is that requests should allow nodes to populate their routing tables. This procedure may seem a simple optimization but it allows a major simplification of the scheme: a node does not have to determine the set of keys it is in charge of. This greatly alleviate the protocol since this set usually depends on the IDs of other nodes which are close to the node ID making it highly dynamic when nodes are inserted and deleted.

To allow this, the lookup procedure allows to find the k nodes with closest ID to a requested key. These k nodes will be in charge of storing an information associated with the key for some period of time. To avoid overcaching, a node stores a key, value association during a time exponentially inversely proportional to the estimated number of nodes having an ID closer to the key. (Notice that key, value associations must be re-published regularly.)

Basically, a node u maintains a bucket B_i for each prefix $u[1,i]$. Each bucket is a cache with IP addresses of alive nodes with ID sharing a common prefix with u: $v[1,i] = u[1,i]$. A lookup consists in iteratively querying a set of nodes for IPs of alive nodes with an ID close to the requested key. Each node gives its bucket with longest common prefix with the key. At each iteration, the set is replaced by the nodes closest to the key until the lookup process comes to a fix point.

A key point of Kademlia is that any node must have an accurate view of the k nodes with closest ID to its own ID. This insures that the lookup procedure effectively converges towards the k nodes with ID closest to the requested key. (This implies a special treatment of the buckets sharing longest common prefix with the node ID.)

3.2 Bucket accuracy in Kademlia

A nice feature of Kademlia resides in bucket refresh through requests. Each time v contacts u performing a request, v may refresh its bucket for u (if u replies) and u may refresh its bucket for v. In other words, both buckets B_i of u and v are refreshed where i is the length of the longest common prefix of u and v. We are going to see that requests indeed mainly refresh buckets with low value of i.

Suppose that each node performs a request once every T seconds on average, each time towards a random key. Now consider how the buckets of some node u are refreshed. The number of nodes which fall in the range of bucket B_i is roughly $n_i = \frac{N}{2^i}$. One of this nodes will contact u at the jth step of the lookup procedure with probability $\frac{\alpha}{n_j 2^j}$ where α is the number of nodes queried in parallel in the lookup procedure, $1/2^j$ is the probability that u shares j bits with the requested key and $1/n_j$ is the probability that u is chosen among all nodes that share j bits with the key. B_i is thus refreshed at rate $O(\log N/2^i)$.

To cope with this problem, Kademlia includes a refresh procedure for buckets that are not sufficiently refreshed by requests. Notice however that buckets where great accuracy is required are the buckets which are less statistically refreshed by requests. On the contrary we are proposing a scheme with only two buckets allowing to achieve maximal accuracy of them.

4 Loose Distributed Hashtable with De-Brujin Tolopy

We now propose a loose topology protocol similar to Kademlia functionning with a topology close to the De-Brujin graph. This protocol is called Broose.

4.1 Distance metric in the De-Brujin topology

To allow a Kademlia like lookup process, we need to define a distance metric between IDs. However, the circular shifting of bits in the De-Brujin topology makes it less intuitive. Suppose we know the distance d in number of hops from a node u to a node v storing information about a key k. The intuition behind the De-Brujin topology is that $u[d+1,n]$, v and k will share a common prefix. The closer u is from v, the longer the prefix will be. We thus define

$$D_d(u,k) = u[d+1,n] \oplus k[1,n-d]$$

as the d-hops distance from u to k. We have chosen to use the xor operation as Kademlia for its simplicity but any definition such that $D_d(u,k)$ gets smaller as $u[d+1,n]$ and $k[1,n-d]$ share a longer prefix would be suitable. More precisely, any distance metric verifying the triangular inequality and such that for any bits b_1 and b_2, $\mathrm{dist}_l(u[2,l]b_1, v[2,l]b_2) = \Theta(\mathrm{dist}_l(u[1,l], v[1,l]))$ for $l \leq n$ would be suitable for defining $D_d(u,k) = \mathrm{dist}_{n-d}(u[d+1,n], k[1,n-d])$. We will prefer the xor metric for which the distance is multiplied by 2 after shifting rather than the euclidian metric for which it may be multiplied by 4. As stated in [4], the xor metric verifies the triangular inequality $u \oplus v + v \oplus w \geq u \oplus w$ since $u \oplus w = (u \oplus v) \oplus (v \oplus w)$ and $a + b \geq a \oplus b$ for $a \geq 0$ and $b \geq 0$.

Notice that this definition of $D_d(u,k)$ is not symmetric conversely to the distance used in Kademlia. This is due to the intrisicly oriented nature of the De-Brujin topology. We will discuss the implications in Section 4.4.

4.2 Routing tables

The main advantage of Broose is to offer an ultimately simple routing table. Each node u simply stores two buckets S and P. S should contain the possible successors of u and is composed of the δ nodes v with ID closest to $u[2,n]$, i.e. those for which $D_1(u,v)$ is minimal ever seen by u and still alive. P should contain the possible predecessors of u and is composed of the δ nodes v for which $D_1(v,u)$ is minimal (those which share a common prefix with either $0u[1,n]$ or $1u[1,n]$). δ is a parameter of the protocol.

P can be split in two sub-buckets P_0 and P_1 according to the first bit of the IDs. Let $|B|$ denote the number node actually stored in a bucket. Notice that it is very unlikely (if IDs are chosen randomly) that $|P_0|$ be much greater or much smaller than $|P_1|$. Indeed, it follows from Chernoff bound[1] that $Pr[|P_0| \leq \frac{1}{4}\delta] \leq e^{-\delta/16}$. Similarly, S can be split in two sub-buckets. Let $u[2,l]$ be the longest common prefix of nodes in S. Define S_0 and S_1 as the subsets of nodes with prefix $u[2,l]0$ and $u[2,l]1$ respectively. Unbalanced bucket avoidance is discussed later in Section 4.6.

4.3 Lookup procedure

To make a lookup for a key k from a node u, one should estimate an upper bound d of the distance in hops from u to a node storing values for k. A lookup set L is initialized with $L_d = \{u\}$ and the following iteration is performed for $i = d-1, \ldots, 0$. Contact all nodes in L_{i+1} and retrieve their S buckets. Let L_i be the union of these. If it contains more than α nodes, keep only the α nodes v that minimize $D_i(v,k)$. α is a parameter of the protocol discussed later. d can be deduced from N since d is roughly $\log N$. See Section 4.6 for hints about estimating $\log N$. A better approach is to continuously estimate the maximal hop distance to keys since half of the keys are at maximal hop distance. Another approach would be to query for key u with increasing d until u finds its own ID in L_0.

To enhance P buckets accuracy, all nodes contacted at phase i should be informed of alive nodes in L_{i-1}. (Alive nodes simply means those that answered here.) (This information can easily be piggybacked in the request.)

4.4 Reverse lookup procedure

A reverse lookup procedure is similarly defined base on the P buckets. A lookup set L is initialized with $L_d = \{u\}$ and the following iteration is performed for $i = d-1, \ldots, 0$. Contact all nodes in L_{i+1} and retrieve their P buckets. Let L_i be the union of these. If it contains more than α nodes, keep only the α nodes v that minimize $\overline{D}_i(v,k)$, where $\overline{D}_i(v,k) = v[1,n-i] \oplus k[i+1,n] = D_i(k,v)$. To enhance S buckets accuracy, all nodes contacted at phase i should be informed of alive nodes in L_{i-1}.

Notice that lookups and reverse lookups are complementary: lookups use the S buckets and contribute to the accuracy of the P buckets whereas reverse lookups use the P buckets and contribute to the accuracy of the S buckets. Lookups and reverse lookups should be indifferently used for performing requests.

4.5 Sketch of proof

To prove that the lookup procedure from u for key k succeeds, we show that L_i always contain the γ closest IDs to $k_i = u[d-i+1,d]k[1,n-i]$ with high probability. This is

[1] Recall that so-called Chernoff bound says that, the number of successes $X(n)$ during n independent Bernoulli trials with probability p of success verifies $Pr[X(n) \leq (1-\epsilon)pn] \leq e^{-\epsilon^2 pn/2}$

achieved by choosing $\alpha = O(\log n)$ and $\delta = O(\log n)$ sufficiently large and relying on bucket accuracy.

Lemma 1 *When node IDs are randomly chosen, the number of nodes at distance less than $c \log N \frac{2^n}{N}$ from a random ID v with respect to the xor metric is $\Theta(\log N)$ w.h.p. (with high probability).*

This lemma is a direct application of Chernoff bounds since the probability that a random ID falls at distance less than $c \log N \frac{2^n}{N}$ from v is $c \log N \frac{1}{N}$. Let X denote the number of nodes at distance less than $c \log N \frac{2^n}{N}$. Its mean value $E[X]$ is thus $m = c \log N$. The Chernoff bounds simply state that $Pr\left[|X - m| \geq \epsilon m\right] \leq \frac{1}{N^{c'}}$ with $c' = e^{-\epsilon^2 c/3}$.

According to Lemma 1, fix some $\gamma = c_1 \log n$ such that the number of nodes at distance at most $x = c \log n$ from a random ID is greater than γ w.h.p..

According to Lemma 1, fix $y = c_2 \log n$ such that the number of nodes at distance at most y is lower than γ w.h.p.. Finally fix $c_3 \geq c_2 + 2c$ and δ such that the number of nodes at distance less or equal to $y + 2x = c_3 \log n$ is lower than δ w.h.p..

Taking an estimated hop distance d sufficiently large allows to assume that $D_d(u, k_0) < x/2$ implying $D_{d-1}(u[2,n], k_1) \leq x$ since $D_{d-1}(u[2,n], k_1) \leq 2D_{d-1}(u[2,n], k_0[2,n]) + 1$. Assuming bucket accuracy, the definition of δ insures that L_{d-1} will contain the γ closest nodes to k_1 with respect to D_{d-1}.

Now for $i < d-1$, let v be one of the γ closest nodes to k_i with respect to D_i. $D_i(v, k_i) < x$ w.h.p., implying $D_{i-1}(v[2,n], k_{i+1}) \leq 2x$ w.h.p.. Any node w among the gamma closest nodes to k_{i+1} verifies $D_{i-1}(w, k_{i+1}) \leq y$ w.h.p. implying that w is at distance at most $y + 2x$ from $v[2,n]$. It is thus in the S bucket of v w.h.p. and will be included in L_{i+1}. This terminates the proof by recurrence of the correctness of the lookup procedure.

Notice that n, the length in bits of IDs, is a trivial upper bound of $\log N$, however 30 or 40 is certainly a practical upper bound of $\log N$ allowing a reasonable choice of $\alpha = 30$ and $\delta = 500$. A similar proof could be made for reverse lookup. Notice that the lookup procedures are quite tolerant to bucket inaccuracies since it is sufficient for the proof to suppose that the union of buckets retrieved from L_i is accurate.

4.6 Avoiding unbalanced ID distribution

One can choose α and δ within a factor sufficiently large for making the probability of failure very low (the probability of failure will tend towards zero as N tends to infinity). However, this choice limits the acceptable unbalancy in the ID distribution. Typically, the smoothness ρ introduced in Section 2.2 must stay within $O(\log n)$.

Naor and Wieder [5] discuss interesting ways of getting better smoothness. (If the choice of new IDs can be easily tuned to get better smoothness, the deletion of nodes is more problematic.) Here, we propose a simple scheme allowing a node to discover that it has chosen a very popular prefix for its ID and that it should leave the network and join it again with a new randomly chosen ID.

For that purpose, a node may regularly ask its successors for their P buckets and its predecessors for their S buckets. Computing the union of all these buckets and keeping the δ nodes closest to its ID with regard to D_0 it gets the bucket R of its "brothers". For each bucket B in S, R, P_0, P_1, it computes the length $l(B)$ of the common prefix of all IDs in B. $l(B)$ can be seen as an estimation of $\log N$. If $l(R)$ is more than $\log \log N$ above $l(S)$, $l(P_0) + 1$, or $l(P_1) + 1$, even after a successive construction of R, then the node detects that the smoothness has reached $O(\log N)$ and that it should change its ID. (However, if R contains less than δ nodes, the network is still too poorly populated to take such a decision.) Notice that the probability of such a situation is low and that a node will rarely change its ID.

Neverless, the Broose network is vulnerable to an adversary inserting nodes with unbalanced IDs in the purpose of breaking the network.

4.7 Discussion

Each request accurately refreshes either the S buckets or the P buckets of nodes closed to the key destination. If destinations are chosen randomly, this happens α times per request interval per node with probability $1/N$ yielding a refresh period twice longer as the mean request interval per node.

To keep key, value association close to the nodes with IDs closest to the key with regard to D_0, these associations should be republished regularly. Typically, if $O(N)$ nodes leave or enter the network during a period of T seconds, they should be republished $O(\gamma / \log N)$ times per T seconds. To avoid hot spots for popular keys, intermediate nodes should cache key, value associations: nodes contacted at stage i of a lookup procedure for a store should keep the association during $T/2^i$ seconds.

The lookup procedure can be accelerated by shifting b bits at a time. The protocol still remains very simple since the number of buckets remains 2. However, their size δ should be multiplied by 2^{b-1}.

4.8 Summary

Thanks to a De-Brujin approach of the Kademlia functioning, Broose offers ultimately simple routing tables for achieving a distributed hashtable peer-to-peer system with refresh frequency of contact information proportional to lookup frequency.

References

[1] P. Fraigniaud and P. Gauron. The content-addressable network d2b. Technical Report LRI 1349, Univ. Paris-Sud, 2003.

[2] M. Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[3] F.T. Leighton. *Introduction to parallel algorithms and architectures*, chapter 3.3. Morgan Kaufmann, 1992.

[4] P. Maymounkov and D. Mazieres. Kademlia: A peerto -peer information system based on the xor metric. In *1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[5] M. Naor and U. Wieder. Novel architectures for p2p applications: the continuous-discrete approach. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures (SPAA)*, 2003.

[6] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, LNCS 2218, pages 329–350, 2001.

[7] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399